03

Modificando la estructura

Transcripción

[00:00] Ya hemos visto que el problema que tenemos es que si nosotros implementamos la solución de implementar el método o los métodos para setear la contraseña e iniciar sesión dentro de funcionario, habilitamos el acceso para el contador, el gerente y el administrador, pero nosotros solamente queremos que gerente y administrador tengan acceso a esos métodos. Contador no es permitido.

[00:30] ¿Cómo podemos solucionar esto? La primera solución que se viene a la mente puede ser digamos crear una segunda clase padre. Por ejemplo puedo hacer un "Ctrl + C", "Ctrl + V", vamos a ponerlo por aquí abajo y se va a llamar FuncionarioAutenticable. Es un nombre un poco raro pero es el que se me viene a la cabeza. Y aquí tenemos a nuestro funcionario autenticable.

[01:07] ¿Entonces cómo podría ser esta solución? Damos control aquí para copiar la flechita, perfecto. ¿Qué gerente extiende de funcionario autenticable, aquí, y a su vez administrador haga lo propio? De esta forma aseguramos pues que solamente estos dos objetos, estas dos clases tengan el método autenticar y ya lo tiramos aquí a funcionario.

[01:37] Vamos a ver en el código con manos a la obra cómo quedaría esto. Aquí le damos un "Ctrl + Z" para que quede como gerente, nuestro código Java quedará así, y vamos a crear una nueva clase y esta clase se va a llamar FuncionarioAutenticable.

[02:02] Esta no va a extender de funcionario, así que le damos finish. Y en FuncionarioAutenticable vamos a añadirle esos métodos, esos de acá, esos atributos, perdón, así que copiamos y pegamos aquí como FuncionarioAutenticable. Ordenamos un poquito el código. Listo. Entonces ahora, todo funcionario autenticable puede tener una clave y va a iniciar sesión, va a poder iniciar sesión.

[02:42] Perfecto. ¿Ahora qué es lo que sigue? Pues hacer que administrador y gerente extiendan de FuncionarioAutenticable. ¿Cómo hacemos esto? Vamos a ir acá arriba y le decimos extends Funcionario, coma, FuncionarioAutenticable. Si se dan cuenta, no me está autocompletando nada, y esto tiene una explicación. ¿Cuál es?

[03:11] En Java no podemos extender de más de una clase. Perfecto. Entonces, el extends sirve para extender, ser hijo de un solo padre. Tú no puedes ser hijo de dos padres a la vez. Tú eres hijo de un solo padre y en este caso tu caso padre es funcional. Java no permite la herencia múltiple. Esto de acá es herencia múltiple y está permitido en lenguajes como C Sharp y me parece que Ruby también lo implementa, pero no es el caso de Java. ¿Por qué?

[03:47] Porque la idea de Java es mantenerlo digamos sintácticamente correcto, que el código manifieste lo que de verdad está ocurriendo en el mundo real, entonces para darle esa robustez no tiene implementada la herencia múltiple, entonces tú no puedes ser hijo de múltiples padres a la vez. ¿Por qué? ¿Cuál sería el mayor problema que podría haber aquí?

[04:15] Si él tiene un método autenticar y él tiene otro método autenticar, ¿de quién vas a heredar tú? Entonces, ahí el código ya comienza a verse un poco feo, ya comienza pues a implementar antipatrones, la escalabilidad incluso se ve limitada en ese punto porque ya comienzan los errores de diseño y digamos en algún momento la bola de nieve se hace demasiado grande pues que ya es imposible manejarla.

[04:46] Y es justamente por eso que Java solamente permite extender de una sola clase. ¿Qué hacemos ahora? Bueno, vamos a quitarle pues el extends de funcionario y lo vamos a dejar con FuncionarioAutenticable. Y como FuncionarioAutenticable, entonces tanto el administrador como el gerente ya deberían poder ejecutar estos métodos. Pero oh, sorpresa. GetBonificacion ya no compila. ¿Por qué? Porque ya no es override.

[05:18] Recuerden esto, si yo elimino esa anotación, compila correctamente. ¿Por qué? Porque él ya va a tener su propio método getBonificacion. Hasta aquí aparentemente funciona, entonces vamos a ver si es verdad. Vamos a borrar este código, vamos a dejarle solamente getBonificacion, borramos este código de acá, listo. Solamente getBonificacion.

[05:51] De misma forma con el gerente, vamos a borrar este código y vamos a decirle que ya no es un funcionario sino es un funcionario autenticable y debería seguir compilando nuestro código. Vamos a sistema interno y vemos claramente que gerente sigue compilando acá en nuestro código, sigue compilando sin ningún problema.

[06:14] Acá dejó de compilar en Test control de bonificacion porque los métodos propios ya de la clase funcionario, como gerente ya no extiende de funcionario, perdió estos métodos, entonces es el primer error que vamos a ver aquí con esa solución. Hemos perdido funcionalidad del funcionario. Ese es el primer error que estamos viendo.

[06:41] Entonces ya desde este punto sabemos que no es el enfoque más ideal. ¿Por qué? Porque hemos sacrificado funcionalidad para obtener otra, y eso es justamente lo que tenemos que evitar. No podemos sacrificar un feature para implementar otro. Sin embargo bueno, aquí no podemos llamar super.getSalario ni nada de eso.

[07:08] Aquí, solamente para que compile, en el caso de gerente lo que yo voy a hacer es hacer que retorne un número equis, digamos 2000, solo por finalidad

de compilación, fines del ejemplo, y aquí voy a crear mi clase pues de test. Vamos a probar si de verdad hace login o de verdad ni siquiera eso podemos hacer.

[07:32] Creamos la clase TestSistemaInterno, perfecto. Entonces le damos finish. Y en nuestro testimonio vamos a crear nuestro main, y en main vamos a darle SistemaInterno, sistema va a ser igual a new SistemaInterno. Punto y coma. Y aquí vamos a crear, digamos un gerente. Gerente1 es igual a new Gerente. Y aquí vamos a crearles un administrador. Admin igual new Administrador.

[08:20] Y ahora vamos a hacer pues que el sistema haga una autenticación del gerente1. Vemos que el código compila correctamente y que nuestro sistema también autentique al administrador, pero ya vimos que no está implementado aumentan para el administrador. ¿Qué podemos hacer acá? Cambiarlo por FuncionarioAutenticable, misma jugada que habíamos hecho anteriormente, solamente que con funcionario.

[08:53] Volvemos para nuestra clase de test y le damos, ahora sí, admin y vemos que él ya acepta administrador, perfecto. Entonces le damos aquí, guardamos todo y le vamos a decir que ejecute este test. Bueno. Hay errores en este proyecto pero no van a interferir en este caso. Dejamos que proceda y nos está dando error en login.

[09:20] A ver, esto es bueno y es malo. Es malo porque no estamos dando login, ¿pero es bueno por qué? Porque está usando el método de nuestro sistema interno, está ejecutando lo que debería ejecutar. ¿Y por qué entonces nos da error en el login? Porque la clave que estamos seteando es una clave diferente a la que está en usuario autenticable.

[09:45] Si yo copio esto y se lo pego aquí, si ejecutamos nuevamente nuestro test, vamos a darle aquí ejecutar, dejamos que proceda, login exitoso. Entonces, ¿la solución funciona? Sí funciona. Ahora, ¿cubre todos los casos de

uso? No. ¿Por qué? Porque está rompiendo funcionalidad existente. Vamos a ver en el siguiente video una solución más ingeniosa aún para ese problema.